# Circular Coinduction

Grigore Roşu[1] and Joseph Goguen[2]

[1] Research Institute for Advanced Computer Science
Automated Software Engineering Group
NASA Ames Research Center
Moffett Field, California, 94035-1000, USA
http://ase.arc.nasa.gov/grosu

[2] Department of Computer Science & Engineering
University of California at San Diego
9500 Gilman Drive, La Jolla, California 92093-0114, USA
http://www-cse.ucsd.edu/users/goguen

**Abstract.** Circular coinduction is a technique for behavioral reasoning that extends cobasis coinduction to specifications with circularities. Because behavioral satisfaction is not recursively enumerable, no algorithm can work for every behavioral statement. However, algorithms using circular coinduction can prove every practical behavioral result that we know. This paper proves the correctness of circular coinduction and some consequences.

## 1 Introduction

Software and hardware systems are growing in complexity, with ever greater possibilities for subtle errors; these can produce significant loss, including human life. Unfortunately, building complex reliable systems is very difficult, due to incompleteness and rapid evolution of requirements, and to difficulties in writing and understanding specifications. One approach is through formal methods, with its two best known branches being model checking and theorem proving; these can reveal inconsistencies, ambiguities and exceptions that could be expensive or impossible to detect otherwise.

This paper is part of our effort to design, implement, evaluate and popularize formal method tools for *behavioral* specification and verification. Our Tatami system [15, 12, 11] and its BOBJ component[1], a behavioral specification and verification language of the OBJ family, use hidden algebra [9, 17]. Although our work is mainly on theorem proving, our results also have implications for model checking; in particular, only one state in a behavioral equivalence class needs to be stored, and instead of hashing all visited states, a behavioral model checker can just check whether a new state is equivalent to one already stored; this is actually how our CCRW [14] (circular coinductive rewriting) algorithm works in its BOBJ implementation.

[1] BOBJ comes from "Behavioral OBJ," where "OBJ" [20] names a family of algebraic programming and specification languages based on parameterized programming and order sorted term rewriting and algebra, possibly enriched with other logics.

Some results in this paper were sketched in [28] and implemented by Kai Lin in BOBJ [11, 13], but this paper gives the first correctness proof for circular coinduction, and for some of its consequences. Although our examples use BOBJ, we do not present BOBJ in detail, but only the features needed for these examples. The latest information on hidden algebra, including the most recent papers, links to related work, and online tutorial material, can be found at `www.cs.ucsd.edu/users/goguen/projs/halg.html`, the hidden algebra homepage.

## 2 Hidden Logic

Today's software systems often follow the "object paradigm," which may be described as having:

1. *objects* with *local state* and operations that modify or observe them, called *methods* and *attributes*, respectively;
2. *classes* that classify objects through an *inheritance hierarchy*; and
3. *concurrent* distributed execution.

Hidden algebra formalizes the object paradigm, but also includes ordinary programs and components. Our behavioral approach is motivated by the fact that cleverly designed systems often fail to satisfy their requirements strictly, but instead only satisfy them *behaviorally*, in the sense of *appearing* to satisfy them under all possible experiments.

Hidden algebra was introduced [9] to give algebraic semantics for the object paradigm, and developed further in [10, 16, 4, 17] among other places. One distinctive feature is a split of sorts into *visible* and *hidden*, where visible sorts are for data and hidden sorts are for objects. A model, or *hidden algebra*, is an abstraction of an implementation, consisting of the possible states, with concrete functions for the attributes and methods, from states to data and to states, respectively (hence, an attribute "observes" states by returning a visible value, while a method modifies states); i.e., a hidden algebra is an algebra that includes a data universe.

*Hidden logic* is the generic name for various logics closely related to hidden algebra, giving sound rules for behavioral reasoning that are easily automated. Following [5], we distinguish two classes of hidden logics, depending on whether the data universe, of "built-ins," is assumed fixed or not. The first versions of hidden logic took the fixed data approach, but we recently noticed that all our inference rules are sound for the larger class of models which need not protect data. Since there are also loose data versions of hidden logic, such as *coherent hidden algebra* [7, 8] and *observational logic* [1, 2, 21], we decided not to restrict our exposition to the fixed data case. Nevertheless, the fixed data hidden logics are often desirable, since real applications use standard booleans and integers rather than arbitrary models; for example, the alternating bit protocol cannot be proved correct unless implementations which do not distinguish 0 from 1 are forbidden.

A detailed presentation of various hidden logics appears in [26] together with relations to many other concepts, a history of hidden algebra with citations, and proofs for some results mentioned but not proved here. We now introduce some of the most basic concepts, assuming familiarity with ordinary many sorted algebra:

**Definition 1.** *Given disjoint sets $V, H$ called* **visible** *and* **hidden sorts**, *a* **loose data hidden $(V,H)$-signature** *is a many sorted $(V \cup H)$-signature. A* **fixed data hidden $(V,H)$-signature** *is a pair $(\Sigma, D)$ where $\Sigma$ is a loose data hidden $(V,H)$-signature and $D$, called the* **data algebra**, *is a many sorted $\Sigma|_V$-algebra. A* **loose data hidden subsignature of** *$\Sigma$ is a loose data hidden $(V,H)$-signature $\Gamma$ with $\Gamma \subseteq \Sigma$ and $\Gamma|_V = \Sigma|_V$. A* **fixed data hidden subsignature of** *$(\Sigma, D)$ is a fixed data hidden $(V,H)$-signature $(\Gamma, D)$ over the same data with $\Gamma \subseteq \Sigma$ and $\Gamma|_V = \Sigma|_V$. The operations in $\Sigma$ with one hidden argument and visible result are called* **attributes**, *those with one hidden argument and hidden result are called* **methods**, *those with two hidden arguments and hidden result are called* **binary methods**, *and those with only (zero or more) visible arguments and hidden result are called* **hidden constants**.

Hereafter we may write "hidden signature" instead of "loose data hidden $(V,H)$-signature" or "fixed data hidden $(V,H)$-signature," since we don't need to distinguish them; also we often write $\Sigma$ for $(\Sigma, D)$.

**Definition 2.** *A* **loose data hidden $\Sigma$-algebra** *$A$ is a $\Sigma$-algebra, and a* **fixed data hidden $(\Sigma, D)$-algebra** *$A$ is a $\Sigma$-algebra $A$ such that $A|_{\Sigma_V} = D$.*

Again, we often write just "hidden algebra." A hidden algebra can be regarded as a "blackbox," the inside of which is not seen, since one is only concerned with its behavior under experiments. Notice that fixed data hidden algebras protect their data: for example, such an implementation of a stack of natural numbers does not corrupt its builtin natural numbers.

We next formalize the notion of "experiment," which informally is an observation of an attribute of a system after it has been perturbed by some methods, using the mathematical concept of context; the symbol $\bullet$ below is a placeholder for the state being experimented upon.

**Definition 3.** *Given a hidden subsignature $\Gamma$ of $\Sigma$, an (appropriate)* **$\Gamma$-context for sort** *$s$ is a term in $T_\Gamma(\{\bullet : s\} \cup Z)$ having exactly one occurrence of a special variable[2] $\bullet$ of sort $s$, where $Z$ is an infinite set of special variables. Let $\mathcal{C}_\Gamma[\bullet : s]$ denote the set of all $\Gamma$-contexts for sort $s$, and $var(c)$ the finite set of variables in a context $c$ except $\bullet$. A $\Gamma$-context with visible result sort is called a* **$\Gamma$-experiment**; *let $\mathcal{E}_\Gamma[\bullet : s]$ denote the set of all $\Gamma$-experiments for sort $s$. When the sort of experiments is important, we use the notation $\mathcal{C}_{\Gamma,s'}[\bullet : s]$ for the $\Gamma$-contexts of sort $s'$ for sort $s$, while $\mathcal{E}_{\Gamma,v}[\bullet : s]$ denotes all the $\Gamma$-experiments of sort $v$ for sort $s$. If $c \in \mathcal{C}_{\Gamma,s'}[\bullet : s]$ and $t \in T_{\Sigma,s}(X)$, then $c[t]$ denotes the term in $T_{\Sigma,s'}(var(c) \cup X)$ obtained from $c$ by substituting $t$ for $\bullet$; formally,*

[2] Special variables are assumed different from any other variables in a given situation.

$c[t] = (\bullet \to t)^*(c)$, *where* $(\bullet \to t)^* : T_\Sigma(var(c) \cup \{\bullet : s\}) \to T_\Sigma(var(c) \cup X)$ *is the unique extension of the map* $(\bullet \to t) : var(c) \cup \{\bullet : s\} \to T_\Sigma(var(c) \cup X)$ *which is identity on* $var(c)$ *and takes* $\bullet : s$ *to* $t$. *Furthermore,* $c$ *generates a map* $A_c : A_s \to [A^{var(c)} \to A_{s'}]$ *on each* $\Sigma$*-algebra* $A$, *defined by* $A_c(a)(\theta) = a^*_\theta(c)$, *where* $a^*_\theta$ *is the unique extension of the map (denoted* $a_\theta$*) that takes* $\bullet$ *to* $a$ *and each* $z \in var(c)$ *to* $\theta(z)$.

The interesting experiments are those of hidden sort, i.e., those with $s \in H$; experiments of visible sort are allowed just to smooth the presentation.

We now define a distinctive feature of hidden logic, behavioral equivalence. Intuitively, two states are behaviorally equivalent iff they cannot be distinguished by any experiment that can be performed on the system.

**Definition 4.** *Given a hidden* $\Sigma$*-algebra* $A$ *and a hidden subsignature* $\Gamma$ *of* $\Sigma$, *the equivalence given by* $a \equiv^\Gamma_\Sigma a'$ *iff* $A_\gamma(a)(\theta) = A_\gamma(a')(\theta)$ *for all* $\Gamma$*-experiments* $\gamma$ *and all maps* $\theta : var(\gamma) \to A$ *is called* $\Gamma$**-behavioral equivalence on** $A$. *We may write* $\equiv$ *instead of* $\equiv^\Gamma_\Sigma$ *when* $\Sigma$ *and* $\Gamma$ *can be inferred from context, and we write* $\equiv_\Sigma$ *when* $\Sigma = \Gamma$. *Given any equivalence* $\sim$ *on* $A$, *an operation* $\sigma$ *in* $\Sigma_{s_1 \ldots s_n, s}$ *is* **congruent for** $\sim$ *iff* $A_\sigma(a_1, \ldots, a_n) \sim A_\sigma(a'_1, \ldots, a'_n)$ *whenever* $a_i \sim a'_i$ *for* $i = 1 \ldots n$. *An operation* $\sigma$ *is* $\Gamma$**-behaviorally congruent for** $A$ *iff it is congruent for* $\equiv^\Gamma_\Sigma$. *We often write just "congruent" instead of "behaviorally congruent"*[3]. *A* **hidden** $\Gamma$**-congruence on** $A$ *is an equivalence on* $A$ *which is the identity on visible sorts and for which each operation in* $\Gamma$ *is congruent.*

The following is the basis for several results below, generalizing a result in [17] to operations that have more than one hidden argument or are not behavioral; see [27, 26] for a proof. Since final algebras do not necessarily exist in this setting, existence of a largest hidden $\Gamma$-congruence does not depend on them, as it does in coalgebra [29, 23, 22].

**Theorem 1.** *Given a hidden subsignature* $\Gamma$ *of* $\Sigma$ *and a hidden* $\Sigma$*-algebra* $A$, *then* $\Gamma$*-behavioral equivalence is the largest hidden* $\Gamma$*-congruence on* $A$.

**Definition 5.** *A hidden* $\Sigma$*-algebra* $A$ $\Gamma$**-behaviorally satisfies** *a* $\Sigma$*-equation* $(\forall X)\ t = t'$, *say* $e$, *iff* $\theta(t) \equiv^\Gamma_\Sigma \theta(t')$ *for each* $\theta : X \to A$; *in this case we write* $A \models^\Gamma_\Sigma e$. *If* $E$ *is a set of* $\Sigma$*-equations, we write* $A \models^\Gamma_\Sigma E$ *if* $A$ $\Gamma$*-behaviorally satisfies each* $\Sigma$*-equation in* $E$.

When $\Sigma$ and $\Gamma$ are clear from context, we may write $\equiv$ and $\models$ instead of $\equiv^\Gamma_\Sigma$ and $\models^\Gamma_\Sigma$, respectively. Also, to simplify the presentation, we only consider unconditional equations here, but the theory also allows conditional equations [17, 18, 26].

**Definition 6.** *A* **behavioral** *(or* **hidden***)* $\Sigma$**-specification** *(or* **-theory***) is a triple* $(\Sigma, \Gamma, E)$ *where* $\Sigma$ *is a hidden signature,* $\Gamma$ *is a hidden subsignature of* $\Sigma$, *and* $E$ *is a set of* $\Sigma$*-equations. The operations in* $\Gamma - \Sigma\restriction_V$ *are called*

[3] A similar notion was given by Padawitz in [24].

**behavioral**. *We usually let $\mathcal{B}$, $\mathcal{B}'$, $\mathcal{B}_1$, etc., denote behavioral specifications. A hidden $\Sigma$-algebra $A$* **behaviorally satisfies** *(or* **is a model of***) a behavioral specification $\mathcal{B} = (\Sigma, \Gamma, E)$ iff $A \models^{\Gamma}_{\Sigma} E$, and in this case we write $A \models \mathcal{B}$; we write $\mathcal{B} \models e$ if $A \models \mathcal{B}$ implies $A \models^{\Gamma}_{\Sigma} e$. An operation $\sigma \in \Sigma$ is* **behaviorally congruent for** *$\mathcal{B}$ iff $\sigma$ is behaviorally congruent for every $A \models \mathcal{B}$.*

The following gives the existence of many congruent operations:

**Proposition 1.** *If $\mathcal{B} = (\Sigma, \Gamma, E)$ is a behavioral specification, then all operations in $\Gamma$, and all hidden constants, are behaviorally congruent for $\mathcal{B}$.*

Of course, depending on $E$, other operations may also be congruent; in fact, our experience is that all operations are congruent in many practical situations.

### 2.1 An Example

We illustrate our concepts an example with infinite streams. These are common in the formal specification and verification of protocols, where they serve as inputs and outputs.

```
bth STREAM is sort Stream .
  protecting NAT .
  op head : Stream -> Nat .
  op tail : Stream -> Stream .
  op _&_  : Nat Stream -> Stream .
  op odd  : Stream -> Stream .
  op even : Stream -> Stream .
  op zip  : Stream Stream -> Stream .

  var N : Nat .  vars S S' : Stream .
  eq head(N & S) = N .                        *** 1
  eq tail(N & S) = S .                        *** 2
  eq head(odd(S)) = head(S) .                 *** 3
  eq tail(odd(S)) = even(tail(S)) .           *** 4
  eq head(even(S)) = head(tail(S)) .          *** 5
  eq tail(even(S)) = even(tail(tail(S))) .    *** 6
  eq head(zip(S,S')) = head(S) .              *** 7
  eq tail(zip(S,S')) = zip(S',tail(S)) .      *** 8
end
```

As usual, `head`, `tail` and `_&_` give the first element, the elements after the first, and place an element at the front of a stream, respectively, while `odd` and `even` give the streams of elements in the odd and even positions, respectively, and `zip` interleaves two streams.

A behavioral theory is declared in BOBJ via the keywords `bth ... end`, with the signature and the equations in between. All sorts declared in a behavioral theory are considered hidden; the visible sorts (here `Nat`) are imported from

some visible (data) specification (here NAT). Also operations are behavioral by default: an operation not intended to be behavioral (which is rather rare in practice) is given the attribute ncong. The models of a behavioral theory are the hidden algebras that behaviorally satisfy all its equations. In our case, the standard model is that of infinite lists of natural numbers, with head and tail as expected (the tail of an infinite list is infinite), and for example, odd(1 2 3 4 5 6 7 8 9 ...) is 1 3 5 7 9 ..., even(1 2 3 4 5 6 7 8 9 ...) is 2 4 6 8 ..., and zip(1 3 5 7 9 ..., 2 4 6 8 ...) is 1 2 3 4 5 6 7 8 9 .... However, there may also be non-standard models; for example, the model with exactly one element in each carrier is valid for any loose data hidden theory.

In this example, $\Gamma$ contains all the operations, because all of them are behavioral by default. Therefore, head(•), tail(•), head(tail(zip(odd(•), :))), are all $\Gamma$-contexts. If $A$ is the standard infinite list model, then two lists are behaviorally equivalent iff they have the same elements in the same order. However, there are models where a stream is an infinite tree, or some other infinite structure, and elements can be behaviorally equivalent but not equal.

One can show that head and tail suffice as behavioral operations, since together they can observe all behaviors of states, and thus define the behavioral equivalence. There are at least two approaches to behavioral operations: one says that $\Gamma$ should contain as few operations as possible, and the other says it contain as many as possible. We advocate the second approach, since it is natural in it to select various subsets, called cobases, that support simple coinduction proofs. Moreover, any operation that is consistent with the intended behavior of a specification, i.e. that preserves the behavioral equivalence, can be added to $\Gamma$ without changing the behavioral equivalence relation [26], and there are convenient congruence criteria to determine whether this is the case, as described in Subsection 4.3.

## 3 Hidden Equational Deduction and Cobasis Coinduction

This section presents our latest version of behavioral deduction, excluding circular coinduction, which is described in the next section, and "explicit coinduction" [17], where the user must provide an explicit relation, since this is difficult to automate. However, we do discuss cobasis coinduction (also called $\Delta$-coinduction), because the relation that it uses can be generated automatically from a cobasis[4]. We expect future work to yield further improvements in mechanizing coinduction.

### 3.1 Hidden Equational Deduction

Ordinary equational deduction is unsound for behavioral satisfaction, because the congruence deduction rule is unsound for operations that are not behaviorally congruent (e.g., for NDSTACK in [18]). The rules below modify the usual equational

[4] Cobases are introduced in the next section.

deduction to account for this. We fix a specification $\mathcal{S} = (\Sigma, U, E)$ and let $\equiv_{Eq}$ be defined on terms by (1)–(5) below.

(1) Reflexivity : $$\frac{}{t \equiv_{Eq} t}$$

(2) Symmetry : $$\frac{t \equiv_{Eq} t'}{t' \equiv_{Eq} t}$$

(3) Transitivity : $$\frac{t \equiv_{Eq} t',\ t' \equiv_{Eq} t''}{t \equiv_{Eq} t''}$$

(4) Substitution : $$\frac{(\forall Y)\ t = t' \in E,\ \ \theta\colon Y \to T_\Sigma(X),\ \ \theta(t_i) \equiv_{Eq} \theta(t_i)}{\theta(t) \equiv_{Eq} \theta(t')}$$

(5) Congruence : $$\left\{ \begin{array}{ll} a) & \dfrac{t \equiv_{Eq} t',\ sort(t,t') \in V}{\sigma(W,t) \equiv_{Eq} \sigma(W,t'),\ \text{for each } \sigma \in Der(\Sigma)} \\ \\ b) & \dfrac{t \equiv_{Eq} t',\ sort(t,t') \in H}{\delta(W,t) \equiv_{Eq} \delta(W,t'),\ \text{for each congruent } \delta \in \Sigma} \end{array} \right\}$$

If $\sigma$ is any derived operation over $\Sigma$ having an argument of sort $s$, and if $t$ is a $\Sigma$-term of sort $s$, then for simplicity we let $\sigma(W,t)$ denote the term obtained from $\sigma$ replacing its argument of sort $s$ by $t$ and using some distinct variables $W$ for the other arguments.

Unlike equational logics, the deduction system above is not complete. In fact, behavioral satisfaction is a $\Pi^0_2$-hard problem [5], so one cannot find an automatic procedure to prove all true statements or disprove all false statements. For the example in Subsection 2.1, one can relatively easily prove

`head(zip(N & S, S'))` $\equiv_{Eq}$ `head(N & zip(S', S))`,
`tail(zip(N & S, S'))` $\equiv_{Eq}$ `tail(N & zip(S', S))`,
`head(zip(odd(S), even(S')))` $\equiv_{Eq}$ `head(S)`, and
`head(tail((zip(odd(S), even(S'))))` $\equiv_{Eq}$ `head(tail(S))`,
`head(tail`$^{100}$`((zip(odd(S), even(S'))))` $\equiv_{Eq}$ `head(tail`$^{100}$`(S))`,

and much more, but it is *not* possible to prove any of the following:

`even(N & S)` $\equiv_{Eq}$ `odd(S)`,
`zip(N & S, S')` $\equiv_{Eq}$ `N & zip(S', S)`,
`zip(odd(S), even(S))` $\equiv_{Eq}$ `S`,
`odd(zip(S, S'))` $\equiv_{Eq}$ `S`.

We will see that some of these can be proved by cobasis coinduction, while others need circular coinduction.

### 3.2 Complete Sets of Observers

A *complete set of observers* [3] is a set of contexts that can "generate" all experiments on a system. The following definition is adapted from [3] to our notation and terminology:

**Definition 7.** *Given a hidden signature $\Gamma$, a* **complete set of observers** *for $\Gamma$ is a set of $\Gamma$-contexts, say $\bot$, such that for each $\Gamma$-experiment $\gamma \in \mathcal{E}_\Gamma[\bullet]$ there is some $\Gamma$-context $\delta \in \bot$ which is a subcontext[5] of $\gamma$.*

This says that every experiment $\gamma$ has the form $\gamma'[\delta]$ for some other "smaller" experiment $\gamma'$ and some $\delta \in \bot$. This notion already has a dual flavor to that of *basis* for structural induction, where for each element $t$ of an abstract data type, there is some other element $t'$ and an operation $\delta$ in the basis such that $t = \delta[t']$. The following provides two easy examples:

**Proposition 2.** *For any $\Gamma$, both $\Gamma$ and $\mathcal{E}_\Gamma[\bullet]$ are complete sets of observers.*

Consider the hidden subsignature $\Gamma$ of the signature of streams in the example in Subsection 2.1 containing only the operations `head` and `tail`. Obviously, $\mathcal{E}_\Gamma[\bullet]$ consists of all the terms of the form `head(tail(...(tail(•))))`, for an arbitrary number of occurrences of `tail`. Then it is easy to see that

$\bot_1 = \{$`head(•)`, `tail(•)`$\} = \Gamma$,
$\bot_2 = \{$`head(•)`, `head(tail(•))`, `tail(tail(•))`$\}$,
$\bot_3 = \{$`head(•)`, `head(tail(•))`, `head(tail(tail(•)))`, `tail(tail(tail(•)))`$\}$,
...
$\bot_\infty = \mathcal{E}_\Gamma[\bullet]$,

are all complete sets of observers for $\Gamma$.

To simplify writing we ambiguously let $\Gamma$ also denote the subset of $\Gamma$-contexts obtained directly, without composition, from the operations in $\Gamma$, such as $\bot_1$ above.

As with induction, where some bases can be better than others for particular proofs, it is possible that some different complete sets of observers are better for different applications. For example, if one defines a stream `blink` by

```
eq head(blink) = 0 .
eq head(tail(blink)) = 1 .
eq tail(tail(blink)) = blink .
```

then it is almost certain that the complete set of observers $\bot_2$ above is better than the others. (The stream `blink` is `0 1 0 1 ...`.)

We do not further develop this topic here, here but refer to [3, 26]. However, we would mention a disadvantage of complete sets of observers, that they do not take into account the whole specification but only its signature. In particular, in the example in Subsection 2.1 where $\Gamma = \Sigma$ contains all the operations, it is pretty cumbersome to find an appropriate complete set of observers.

[5] That is, a subterm; notice that $\delta$ necessarily contains the variable • from $\gamma$.

### 3.3 Strong Cobases

The complete formal definition of a strong cobasis is quite technical and not relevant to our work, so we skip it. Intuitively, a strong cobasis is a complete set of observers that takes into account the *equations* of a specification in showing that "for each $\Gamma$-experiment $\gamma$ there is some context $\delta \in \Delta$ which is a subcontext of $\gamma$".

In the example of streams with $\Gamma = \Sigma$, one can tediously prove by induction on the structure of contexts that any experiment is equal to an experiment containing only `head` and `tail` operations, so all the complete sets of observers $\Delta_1$, $\Delta_2$, ..., $\Delta_\infty$ for $\Gamma = \{\texttt{head}, \texttt{tail}\}$ in the previous subsection really are strong cobases for the original specification of streams. A less intuitive strong cobasis for streams is $\{\texttt{head}, \texttt{odd}, \texttt{even}\}$, and one can also tediously show that any experiment is equivalent to an experiment containing only `head`, `odd` and `even`. Intuitively, this is because the three operations can "observe" any element in a stream. For example, `head(even(odd(odd(S))))` observes the fifth element of `S`, while the experiment `head(even(even(odd(even(odd(S))))))` observes the 27th element:

$$\begin{aligned}
\texttt{S} &= a_1\ a_2\ a_3\ a_4\ a_5\ a_6\ a_7\ a_8\ a_9\ \cdots \\
\texttt{odd(S)} &= a_1\ a_3\ a_5\ a_7\ a_9\ a_{11}\ a_{13}\ a_{15}\ \cdots \\
\texttt{even(odd(S))} &= a_3\ a_7\ a_{11}\ a_{15}\ a_{19}\ a_{23}\ a_{27}\ a_{31}\ \cdots \\
\texttt{odd(even(odd(S)))} &= a_3\ a_{11}\ a_{19}\ a_{27}\ a_{35}\ a_{43}\ a_{51}\ a_{59}\ \cdots \\
\texttt{even(odd(even(odd(S))))} &= a_{11}\ a_{27}\ a_{43}\ a_{59}\ \cdots \\
\texttt{even(even(odd(even(odd(S)))))} &= a_{27}\ a_{59}\ \cdots \\
\texttt{head(even(even(odd(even(odd(S))))))} &= a_{27}
\end{aligned}$$

There are situations where the latter cobasis is better than the standard one: see [26] for a detailed presentation of strong cobases, together with more elegant proofs that the above are all strong cobases, and a proof that any complete set of observers is a strong cobasis.

### 3.4 General Cobases

Our general notion of cobasis (see also [18, 19, 25]) is as follows:

**Definition 8.** *If $\mathcal{B}' = (\Sigma', \Gamma', E')$ is a conservative extension of $\mathcal{B} = (\Sigma, \Gamma, E)$ and if $\Delta \subseteq \Sigma'$, then $\Delta$ is a* **cobasis for** *$\mathcal{B}$ iff for all hidden sorted terms $t, t' \in T_{\Sigma,h}(X)$, if $\mathcal{B}' \models (\forall W, X)\ \delta(W, t) = \delta(W, t')$ for all appropriate $\delta \in \Delta$ then $\mathcal{B} \models (\forall X)\ t = t'$.*

The following is a key first step toward automation of coinduction; it was first proved in [27]:

**Theorem 2.** *Every strong cobasis is a cobasis.*

To ease presentation, from now on suppose that $\Delta$ is a cobasis of $\mathcal{B}$ with $\mathcal{B}' = (Der(\Sigma), \Gamma, E)$ and $\Delta \subseteq Der(\Gamma)$, where $Der(\Sigma)$ denotes the set of all $\Sigma$-derived operations.

### 3.5 $\Delta$-Coinduction

Once a cobasis is available, coinduction can be applied automatically. Let $\equiv_{Eq,\Delta}$ be the relation generated[6] by rules (1)–(5) in Subsection 3.1, plus

$$\text{(6) } \Delta\text{-Coinduction: } \frac{\delta(W,t) \equiv_{Eq,\Delta} \delta(W,t') \text{ for all appropriate } \delta \in \Delta}{t \equiv_{Eq,\Delta} t'}$$

The following is immediate from the definition of cobasis:

**Proposition 3.** $\equiv_{Eq} \subseteq \equiv_{Eq,\Delta} \subseteq \equiv$.

Thus, to prove that terms $t, t'$ are behaviorally equivalent, it suffices to show that $t \equiv_{Eq,\Delta} t'$. In particular, in our stream example, where $\Delta = \{\texttt{head}(\bullet), \texttt{tail}(\bullet)\}$ is a cobase, one can immediately prove by $\Delta$-coinduction and equational reasoning that

$$\texttt{zip(N\&S, S')} \equiv_{Eq,\Delta} \texttt{N\&zip(S', S)},$$

by showing that `head` applied either term is `N`, and that `tail` applied to either term is `zip(S', S)`. One can also prove $\texttt{even(N\&S)} \equiv_{Eq,\Delta} \texttt{odd(S)}$, and many other similar behavioral properties.

## 4 Circular Coinduction

This section gives an inference rule for behavioral reasoning, called circular coinduction, since it handles some examples with circularities (i.e., infinite recursions) that could not be handled by previous rules here (or in [27, 18, 19, 25]); we may also call it circular $\Delta$-coinduction or $\Delta^c$-coinduction.

After exploring how to prove the congruence of operations in [27] (see also [26] and Subsection 4.3 below), we became convinced that this does not differ essentially from proving other behavioral properties, except perhaps that it is usually easier. Also certain "coinductive patterns" that appeared in specifying operations inspired a congruence criterion that could automatically decide whether an operation is congruent [27, 26]; moreover, this criterion followed from the $\Delta$-coinduction rule and was strong enough for all proofs we knew at that time. But the fact that the congruence of `zip` in Subsection 2.1 (in the context in which only `head` and `tail` are declared behavioral) didn't follow by that criterion, suggested that more powerful deduction rules were needed.

Bidoit and Hennicker [3] gave a general congruence criterion from which the congruence of `zip` followed easily. Influenced by the relationship between $\Delta$-coinduction and the congruence criterion in [27], we sought a general inference rule from which the criterion in [3] would follow as naturally as our criterion in [27] followed from $\Delta$-coinduction, and which could prove behavioral properties not provable by $\Delta$-coinduction. The result of this search was circular $\Delta$-coinduction, as presented in this section and implemented in BOBJ [14].

[6] Strictly speaking, $\equiv_{Eq}$ should be replaced by $\equiv_{Eq,\Delta}$ in rules (1)–(5).

### 4.1 Limitations of $\Delta$-Coinduction

We first give some examples where the six rules generating the relation $\equiv_{Eq,\Delta}$ are not enough to prove certain simple properties, which however can be easily proved by circular $\Delta$-coinduction.

Suppose one wants to prove that `zip(odd(S), even(S)) ≡ S` holds in the behavioral specification of Subsection 2.1. Let us choose the standard (strong) cobasis $\Delta = \{\texttt{head}(\bullet), \texttt{tail}(\bullet)\}$. For $\Delta$-coinduction, one has to prove that $\texttt{head}(\texttt{zip}(\texttt{odd}(\texttt{S}), \texttt{even}(\texttt{S}))) \equiv_{Eq,\Delta} \texttt{head}(\texttt{S})$, which follows by equational deduction, and that $\texttt{tail}(\texttt{zip}(\texttt{odd}(\texttt{S}), \texttt{even}(\texttt{S}))) \equiv_{Eq,\Delta} \texttt{tail}(\texttt{S})$, which reduces to $\texttt{zip}(\texttt{even}(\texttt{S}), \texttt{even}(\texttt{tail}(\texttt{S}))) \equiv_{Eq,\Delta} \texttt{tail}(\texttt{S})$. By $\Delta$-coinduction, one similarly generates two other subgoals, namely $\texttt{head}(\texttt{zip}(\texttt{even}(\texttt{S}), \texttt{even}(\texttt{tail}(\texttt{S})))) \equiv_{Eq,\Delta} \texttt{head}(\texttt{tail}(\texttt{S}))$, which is easy, and $\texttt{tail}(\texttt{zip}(\texttt{even}(\texttt{S}), \texttt{even}(\texttt{tail}(\texttt{S})))) \equiv_{Eq,\Delta} \texttt{tail}(\texttt{tail}(\texttt{S}))$, which reduces to $\texttt{zip}(\texttt{even}(\texttt{tail}(\texttt{S})), \texttt{even}(\texttt{tail}(\texttt{tail}(\texttt{S})))) \equiv_{Eq,\Delta} \texttt{tail}(\texttt{tail}(\texttt{S}))$. Since the last subgoal is nothing but the previous (hidden) one where S is replaced by `tail(S)`, this procedure will loop forever, and thus does not work. But circular coinduction will detect this circularity and terminate, declaring the initial goal proved. Before we discovered and implemented circular coinduction, BOBJ either froze or reported a "segmentation fault" when asked to automatically prove such properties. We encourage the interested reader try to prove $\texttt{odd}(\texttt{zip}(\texttt{S}, \texttt{S}')) \equiv_{Eq,\Delta} \texttt{S}$ with basis coinduction, and to discover another seemingly hopeless circularity there.

### 4.2 Circular $\Delta$-Coinduction

Let $\mathcal{B} = (\Sigma, \Gamma, E)$ be a fixed behavioral specification for this subsection. To ease the presentation, suppose that $\Delta$ is a complete set of observers. Technically speaking, $\Delta$ can be a strict cobasis but the proofs are slightly more complicated; although we haven't yet proved the correctness of circular coinduction for general cobases, this doesn't seem to have any practical relevance, since all the concrete cobases we know are either complete sets of observers or are strong cobases. We consider all equations to be quantified by exactly the variables that occur in their two terms, and omit them whenever possible; we also write $t \equiv t'$ instead of $\mathcal{B} \models (\forall X)\ t = t'$.

**Definition 9.** *Substitutions $\theta, \theta' : X \to T_\Gamma(Y)$ are* **behaviorally equivalent**, *written $\theta \equiv \theta'$, iff $\theta(x) \equiv \theta'(x)$ for every $x \in X$. Terms $t$ and $t'$ are* **strongly behaviorally equivalent**, *written $t \mathrel{\widetilde{\equiv}} t'$, iff for any $\mathcal{B}$-algebra $A$ and any $\tau_1, \tau_2 : X \to A$ with $\tau_1(x) \equiv^{\Gamma}_{\Sigma} \tau_2(x)$ for each $x \in X$, $\tau_1(t) \equiv^{\Gamma}_{\Sigma} \tau_2(t')$.*

Notice that $\mathrel{\widetilde{\equiv}}$ is symmetric and transitive but may not be reflexive, since, for example, terms of the form $\sigma(x_1, \ldots, x_n)$ are not strongly equivalent to any term if $\sigma$ is not congruent (see also 5 of Proposition 4).

**Proposition 4.** *The following hold:*

1. $t = t'$ *implies* $t \tilde{=} t'$;
2. $t \tilde{=} u$ *iff* $t = u$, *whenever* $u$ *is a* $\Gamma$*-term*[7];
3. $t \tilde{=} t'$ *iff* $\gamma[t] \tilde{=} \gamma[t']$ *for all appropriate* $\Gamma$*-experiments* $\gamma$;
4. $t \tilde{=} t'$ *and* $\theta \equiv \theta'$ *imply* $\theta(t) \tilde{=} \theta'(t')$;
5. $\sigma$ *is congruent iff* $\sigma(x_1, \ldots, x_n) \tilde{=} \sigma(x_1, \ldots, x_n)$.

*Proof.* 1. This is straightforward since one can take $\tau_1 = \tau_2$ in Definition 9.

2. If $t \tilde{=} u$ then $t = u$ by 1. Now suppose that $t \equiv u$ and let $\tau_1, \tau_2$ be like in Definition 9. Since $u$ contains only congruent operations, then one can easily show by structural induction that $\tau_1(u) \equiv^{\Gamma}_{\Sigma} \tau_2(u)$. On the other hand, since $\tau_1(t) \equiv^{\Gamma}_{\Sigma} \tau_1(u)$ and $\tau_2(t) \equiv^{\Gamma}_{\Sigma} \tau_2(u)$, it follows that $t \tilde{=} u$.

3. Suppose that $t \tilde{=} t'$, that $\gamma$ is a $\Gamma$-experiment and that $\tau_1, \tau_2 : var(t, t') \cup var(\gamma) \to A$ are maps as in Definition 9. It is immediate that $\tau_1(t) \equiv^{\Gamma}_{\Sigma} \tau_2(t')$. Since $\gamma$ contains only congruent operations, it can be easily seen that $\tau_1(\gamma[t]) = A_\gamma(\tau_1(t))(\tau_1) = A_\gamma(\tau_2(t'))(\tau_1) = A_\gamma(\tau_2(t'))(\tau_2) = \tau_2(\gamma[t'])$. Conversely, suppose that $\gamma[t] \tilde{=} \gamma[t']$ for all appropriate $\Gamma$-experiments $\gamma$, and let $\tau_1, \tau_2 : var(t, t') \to A$ be two maps as in Definition 9. It suffices to show that for any $\Gamma$-experiment $\gamma$, $A_\gamma(\tau_1(t)) = A_\gamma(\tau_2(t'))$ as functions in $[(var(\gamma) \to A) \to A]$. Notice that giving a function in $[var(\gamma) \to A]$ implies extending $\tau_1, \tau_2$ to functions $var(t, t') \cup var(\gamma) \to A$, in which case, $A_\gamma(\tau_1(t)) = \tau_1(\gamma[t]) = A_\gamma(\tau_2(t')) = \tau_2(\gamma[t'])$.

4. this follows by noticing that for any $\tau_1, \tau_2 : Y \to A$ with $\tau_1(y) \equiv^{\Gamma}_{\Sigma} \tau_2(y)$, and any $\theta, \theta' : X \to T_\Gamma(Y)$ with $\theta \equiv \theta'$, it is the case that the maps $\theta; \tau_1, \theta'; \tau_2 : X \to A$ also satisfy the property that $(\theta; \tau_1)(x) \equiv^{\Gamma}_{\Sigma} (\theta'; \tau_2)(x)$ for each $x \in X$.

5. $\sigma$ is congruent iff $A_\sigma(a_1, \ldots, a_n) \equiv A_\sigma(a'_1, \ldots, a'_n)$ for any $a_1, a'_1, \ldots, a_n, a'_n$ with $a_1 \equiv a'_1, \ldots, a_n \equiv a'_n$ iff $\tau_1(\sigma(x_1, \ldots, x_n)) \equiv \tau_2(\sigma(x_1, \ldots, x_n))$ for $\tau_1(x_i) = a_i$ and $\tau_2(x_i) = a'_i$ for all $1 \leq i \leq n$ iff $\sigma(x_1, \ldots, x_n) \tilde{=} \sigma(x_1, \ldots, x_n)$.

For the rest of the section, we assume some well-founded partial order $<$ on $\Gamma$-contexts which is preserved by the operations in $\Gamma$. For example, one such order is the depth of contexts.

**Definition 10.** *Terms* $t$ *and* $t'$ *are* $\Delta^{\tilde{}}$**-coinductively equivalent** *iff for each appropriate* $\delta \in \Delta$, *either* $\delta(W, t) \equiv \delta(W, t') \equiv u$ *for some* $\Gamma$*-term* $u$, *or* $\delta(W, t) \equiv \theta(c[t])$ *and* $\delta(W, t') \equiv \theta'(c[t'])$ *for some* $\theta \equiv \theta'$ *and* $c < \delta$.

**Theorem 3.** *If* $t$ *and* $t'$ *are* $\Delta^{\tilde{}}$*-coinductively equivalent then* $t \tilde{=} t'$.

*Proof.* We first show by well-founded induction that for every appropriate experiment $\gamma$, $\gamma[t] \tilde{=} \gamma[t']$. Let $\gamma$ be any experiment and assume that $\gamma'[t] \tilde{=} \gamma'[t']$ for all experiments $\gamma' < \gamma$. Since $\Delta$ is a complete set of observers, there is some

[7] We write "$\Gamma$-terms" for simplicity, but the result holds for all terms built with congruent operations.

experiment $\gamma''$ such that $\gamma = \gamma''[\delta]$ for some $\delta \in \Delta$. If there is some $\Gamma$-term $u$ such that $\delta(W,t) \equiv \delta(W,t') = u$ then $\gamma[t] = \gamma[t'] \equiv \gamma''[u]$ and $\gamma''[u]$ is a $\Gamma$-term, so by 2 of Proposition 4, $\gamma[t] \stackrel{\cdot}{\equiv} \gamma[t']$. On the other hand, if $\delta(W,t) \equiv \theta(c[t])$ and $\delta(W,t') \equiv \theta'(c[t'])$ for some $\theta \equiv \theta'$ and $c < \delta$, then since the variables appearing in contexts are assumed to be always different from the other variables, one gets that $\gamma[t] = \theta(\gamma''[c[t]])$ and $\gamma[t'] = \theta'(\gamma''[c[t']])$, and so by the induction hypothesis for $\gamma' = \gamma''[c] < \gamma''[\delta] = \gamma$ and 4 of Proposition 4, $\gamma[t] \stackrel{\cdot}{\equiv} \gamma[t']$. The rest follows by 3 of Proposition 4.

Therefore we can add a new inference rule. Since in most cases $\theta = \theta'$, we let $\equiv_{Eq,\Delta}$ be the relation generated[8] by the rules (1)-(6) in Subsections 3.1 and 3.5 and the following:

$$(7)\ \Delta^{\cdot}\text{-Coinduction}: \quad \frac{\begin{array}{c}(\delta(W,t) \equiv_{Eq,\Delta} u \equiv_{Eq,\Delta} \delta(W,t') \\ \text{where } u \text{ is some } \Gamma\text{-term}) \text{ or} \\ (\delta(W,t) \equiv_{Eq,\Delta} \theta(c[t]) \text{ and } \delta(W,t') \equiv_{Eq,\Delta} \theta(c[t']) \\ \text{for some } c < \delta) \text{ for all appropriate } \delta \in \Delta\end{array}}{t \equiv_{Eq,\Delta} t'}$$

In order to prove that $t \equiv t'$, one can prove now that $t \equiv_{Eq,\Delta} t'$. For example, to prove that `zip(odd(S), even(S)) ≡ S`, the property that sent $\Delta$-coinduction into an infinite loop in Subsection 4.1, one can first prove that `zip(even(S), even(tail(S))) ≡`$_{Eq,\Delta}$ `tail(S)` by `{head,tail}`$^{\cdot}$-coinduction (if $\delta$ is `head` then we are in the first case of (7) and if $\delta$ is `tail` then we are in the second case of (7) with $c = \bullet$ and $\theta$(`S`) = `tail(S)`), and then to prove by `{head,tail}`-coinduction the original behavioural equality as in Subsection 4.1. We suggest the reader prove that `odd(zip(S,S')) ≡ S` also by `{head,tail}`$^{\cdot}$-coinduction and then prove both statements by `{head,odd,even}`-coinduction.

BOBJ implements circular coinductive rewriting [14, 13], an algorithm that combines the coinduction inference rules presented in this paper with behavioral rewriting, an adaptation of term rewriting to our behavioral equational deduction system; this can automatically prove all the reasonable statements that we know, including all those mentioned in this paper, and all those that we tried from examples previously done by CoClam [6] using complex heuristics, but of course new inference rules may be needed for more exotic examples.

### 4.3 Congruence Criteria

The simplest way to find a cobasis for a behavioral specification is to guess one and then to show that all the other operations are behaviorally congruent

[8] Strictly speaking, $\equiv_{Eq}$ in rules (1)-(5) and $\equiv_{Eq,\Delta}$ is rule (6) should be replaced by $\equiv^{\cdot}_{Eq,\Delta}$.

for a specification having the same equations and operations as the original specification but only the guessed operations declared as behavioral (see [18, 26] for more detail). Since one of our major goals is to automate the process of behavioral deduction in BOBJ, the problem of automatic detection of cobases plays a crucial role. BOBJ implements a heuristic that works well in practical situations, and is based on the following criteria, which follow from Theorem 3. The first congruence criterion, which we will call the **BH criterion**, is the essence of that in [3]:

**Corollary 1.** *Given a complete set $\Delta$ of observers and some $\sigma \in \Sigma$ such that for each $\delta \in \Delta$, either $\delta[\sigma(x_1, \ldots, x_n)] \equiv u$ for some $\Gamma$-term $u$, or else $\delta[\sigma(x_1, \ldots, x_n)] = c[\sigma(t, \ldots, t_n)]$ for some $\Gamma$-terms $t, \ldots, t_n$ and $c < \delta$, then $\sigma$ is congruent.*

*Proof.* Theorem 3 with $t = t' = \sigma(x_1, \ldots, x_n)$ and $\theta = \theta'$ with $\theta(x_i) = t_i$ for all $1 \leq i \leq n$, gives $\sigma(x_1, \ldots, x_n) \mathrel{\widetilde{\equiv}} \sigma(x_1, \ldots, x_n)$. Then 5 of Proposition 4 gives congruence of $\sigma$.

The following simpler but common congruence criterion, which we here call the **RG criterion**, was presented in [27] together with the suggestion that it could be easily implemented in a system like CafeOBJ:

**Corollary 2.** *Given an operation $\sigma \in \Sigma$ such that for each $\delta \in \Gamma$, if the equation $\delta[\sigma(x_1, \ldots, x_n)] = u$ for some $\Gamma$-term $u$ is in $E$, then $\sigma$ is congruent.*

*Proof.* This is the special case of the BH criterion where $\Delta = \Gamma$ and there is no circularity (i.e., recurrence) in the definition of $\sigma$.

## References


1. Gilles Bernot, Michel Bidoit, and Teodor Knapik. Observational specifications and the indistinguishability assumption. *Theoretical Computer Science*, 139(1-2):275–314, 1995. Submitted in 1992.
2. Michel Bidoit and Rolf Hennicker. Behavioral theories and the proof of behavioral properties. *Theoretical Computer Science*, 165(1):3–55, 1996.
3. Michel Bidoit and Rolf Hennicker. Observer complete definitions are behaviourally coherent. In Kokichi Futatsugi, Joseph Goguen, and José Meseguer, editors, *OBJ/CafeOBJ/Maude at Formal Methods '99*, pages 83–94. Theta, 1999. Proceedings of a workshop held in Toulouse, France, 20th and 22nd September 1999.
4. Rod Burstall and Răzvan Diaconescu. Hiding and behaviour: an institutional approach. In A. William Roscoe, editor, *A Classical Mind: Essays in Honour of C.A.R. Hoare*, pages 75–92. Prentice-Hall, 1994. Also Technical Report ECS-LFCS-8892-253, Laboratory for Foundations of Computer Science, University of Edinburgh, 1992.
5. Samuel Buss and Grigore Roşu. Incompleteness of behavioral logics. In Horst Reichel, editor, *Proceedings of Coalgebraic Methods in Computer Science (CMCS'00), Berlin, Germany, March 2000*, volume 33 of *Electronic Notes in Theoretical Computer Science*, pages 61–79. Elsevier Science, 2000.

6. Louise Dennis, Alan Bundy, and Ian Green. Using a generalisation critic to find bisimulations for coinductive proofs. In William McCune, editor, *Proceedings of the 14th Conference on Automated Deduction*, volume 1249 of *Lecture Notes in Artificial Intelligence*, pages 276–290. Springer, 1996.
7. Răzvan Diaconescu and Kokichi Futatsugi. *CafeOBJ Report. The Language, Proof Techniques, and Methodologies for Object-Oriented Algebraic Specification*. World Scientific, 1998. AMAST Series in Computing, volume 6.
8. Răzvan Diaconescu and Kokichi Futatsugi. Behavioral coherence in object-oriented algebraic specification. *Journal of Universal Computer Science*, 6(1):74–96, 2000. Also Japan Advanced Institute for Science and Technology Technical Report number IS-RR-98-0017F, 1998.
9. Joseph Goguen. Types as theories. In George Michael Reed, Andrew William Roscoe, and Ralph F. Wachter, editors, *Topology and Category Theory in Computer Science*, pages 357–390. Oxford, 1991. Proceedings of a Conference held at Oxford, June 1989.
10. Joseph Goguen and Răzvan Diaconescu. Towards an algebraic semantics for the object paradigm. In Hartmut Ehrig and Fernando Orejas, editors, *Proceedings, Tenth Workshop on Abstract Data Types*, pages 1–29. Springer, 1994. Lecture Notes in Computer Science, Volume 785.
11. Joseph Goguen, Kai Lin, Akira Mori, Grigore Roşu, and Akiyoshi Sato. Distributed cooperative formal methods tools. In *Proceedings, International Conference on Automated Software Engineering (ASE'97)*, pages 55–62. Institute of Electrical and Electronics Engineers, 1997. Lake Tahoe, Nevada, 3-5 November 1997.
12. Joseph Goguen, Kai Lin, Akira Mori, Grigore Roşu, and Akiyoshi Sato. Tools for distributed cooperative design and validation. In *Proceedings, CafeOBJ Symposium*. Japan Advanced Institute for Science and Technology, 1998. Numazu, Japan, April 1998.
13. Joseph Goguen, Kai Lin, and Grigore Roşu. Behavioral and coinductive rewriting. In *Proceedings, Rewriting Logic Workshop, 2000*, pages 1–22. JAIST, 2000. Held in Kanazawa, Japan, September 2000.
14. Joseph Goguen, Kai Lin, and Grigore Roşu. Circular coinductive rewriting. In *Proceedings, Automated Software Engineering '00*, pages 123–131. IEEE, 2000. (Grenoble, France).
15. Joseph Goguen, Kai Lin, Grigore Roşu, Akira Mori, and Bogdan Warinschi. An overview of the Tatami project. In *Cafe: An Industrial-Strength Algebraic Formal Method*, pages 61–78. Elsevier, 2000.
16. Joseph Goguen and Grant Malcolm. Proof of correctness of object representation. In A. William Roscoe, editor, *A Classical Mind: Essays in Honour of C.A.R. Hoare*, pages 119–142. Prentice-Hall, 1994.
17. Joseph Goguen and Grant Malcolm. A hidden agenda. *Theoretical Computer Science*, 245(1):55–101, August 2000. Also UCSD Dept. Computer Science & Eng. Technical Report CS97-538, May 1997.
18. Joseph Goguen and Grigore Roşu. Hiding more of hidden algebra. In *FM'99 – Formal Methods*, pages 1704–1719. Springer, 1999. Lecture Notes in Computer Sciences, Volume 1709, Proceedings of World Congress on Formal Methods, Toulouse, France.
19. Joseph Goguen and Grigore Roşu. A protocol for distributed cooperative work. In Gheorghe Ştefănescu, editor, *Proceedings of Workshop on Distributed Systems 1999 (WDS'99), Iasi, Romania, 2 September 1999*, volume 28 of *Electronic Notes in Theoretical Computer Science*, pages 1–22. Elsevier Science, 1999.

20. Joseph Goguen, Timothy Winkler, José Meseguer, Kokichi Futatsugi, and Jean-Pierre Jouannaud. Introducing OBJ. In Joseph Goguen and Grant Malcolm, editors, *Software Engineering with OBJ: algebraic specification in action*, pages 3–167. Kluwer, 2000.
21. Rolf Hennicker and Michel Bidoit. Observational logic. In *Algebraic Methodology and Software Technology (AMAST'98)*, volume 1548 of *Lecture Notes in Computer Science*, pages 263–277. Springer, 1999.
22. Bart Jacobs. Mongruences and cofree coalgebras. In Maurice Nivat, editor, *Algebraic Methodology and Software Technology (AMAST95)*, pages 245–260. Springer, 1995. Lecture Notes in Computer Science, Volume 936.
23. Bart Jacobs and Jan Rutten. A tutorial on (co)algebras and (co)induction. *Bulletin of the European Association for Theoretical Computer Science*, 62:222–259, 1997.
24. Peter Padawitz. Towards the one-tiered design of data types and transition systems. In *Proceedings, WADT'97*, volume 1376 of *Lecture Notes in Computer Science*, pages 365–380. Springer, 1998.
25. Grigore Roşu. Behavioral coinductive rewriting. In Kokichi Futatsugi, Joseph Goguen, and José Meseguer, editors, *OBJ/CafeOBJ/Maude at Formal Methods '99*, pages 179–196. Theta, 1999. Proceedings of a workshop held in Toulouse, France, 20th and 22nd September 1999.
26. Grigore Roşu. *Hidden Logic*. PhD thesis, University of California at San Diego, 2000.
27. Grigore Roşu and Joseph Goguen. Hidden congruent deduction. In Ricardo Caferra and Gernot Salzer, editors, *Automated Deduction in Classical and Non-Classical Logics*, pages 252–267. Springer, 2000. Lecture Notes in Artificial Intelligence, Volume 1761; papers from First Order Theorem Proving '98 (FTP'98), Vienna, November 1998.
28. Grigore Roşu and Joseph Goguen. Circular coinduction. Technical Report CSE2000-0647, University of California at San Diego, February 2000. Written October 1999.
29. J.J.M.M. Rutten. Universal coalgebra: a theory of systems. Technical Report CS-R9652, CWI, 1996.